Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

MTG_CD: Multi-scale learnable transformation graph for fault classification and diagnosis in microservices

Juan Chen[1], Rui Zhang[1], Peng Chen[1*], Jianhua Ren[2], Zongling Wu[3], Yang Wang[1], Xi Li[1] and Ling Xiong[1]

Abstract

The rapid advancement of microservice architecture in the cloud has led to the necessity of effectively detecting, classifying, and diagnosing run failures in microservice applications. Due to the high dynamics of cloud environments and the complex dependencies between microservices, it is challenging to achieve robust real-time system fault identification. This paper proposes an interpretable fault diagnosis framework tailored for microservice architecture, namely Multi-scale Learnable Transformation Graph for Fault Classification and Diagnosis(MTG_CD). Firstly, we employ multi-scale neural transformation and graph structure adjacency matrix learning to enhance data diversity while extracting temporal-structural features from system monitoring metrics Secondly, a graph convolutional network (GCN) is utilized to fuse the extracted temporal-structural features in a multi-feature modeling approach, which helps to improve the accuracy of anomaly detection. To identify the root cause of system faults, we finally conduct a coarse-grained level diagnosis and exploration after obtaining the results of classifying the fault data. We evaluate the performance of MTG_CD on the microservice benchmark SockShop, demonstrating its superiority over several baseline methods in detecting CPU usage overhead, memory leak, and network delay faults. The average macro F1 score improves by 14.05%.

Keywords Microservice architecture, Neural transformation, Graph convolution network, Fault diagnosis, Fault detection

Introduction

In recent years, with the popularization of cloud computing and distributed systems, large monolithic services have been gradually rearchitected into finer-grained modules, which combine hundreds or even thousands of loosely-coupled microservices [1]. This transformation involves breaking down single-tenant services into smaller, more concentrated microservices. The microservices architecture offers several advantages that make it a powerful approach, including simplifying deployment of applications and improving the efficiency and flexibility of resource provisioning.

The complexity and dynamics of the deployment microservices environment, along with the complex connection between microservices, can lead to the propagation of system faults when a micro-service fails. For example, as shown in Fig. 1, when a system fault occurs in the Shipping service, it then propagates to the Order service, and finally affects to the Front-end service. The depth of the red represents the severity of the fault superposition. This propagation can result in cascading effects, where the failure of one micro-service can cause issues in other connected microservices, potentially leading to a

*Correspondence:
Peng Chen
chenpeng@mail.xhu.edu.cn
[1] School of Computer and Software Engineering, Xihua University, Chengdu, China
[2] West China Second University hospital, Sichuan University, Chengdu, China
[3] School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China

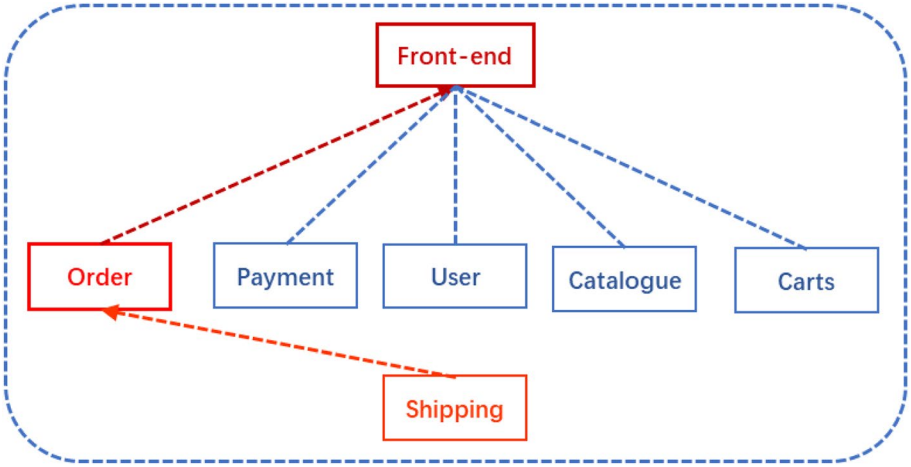

Fig. 1 An example of microservices system fault propagation

complete system failure. Therefore, it is crucial to quickly identify potential issues in microservices before they can cause widespread disruption, which helps to guide the fault-tolerant and elastic scheduling, so as to alleviate the impact of system faults and ensure continuous service availability.

Identifying and diagnosing faults in microservice-based systems poses unique challenges [2], primarily due to the inherent complexity and dynamic nature of microservices in four aspects: nodes, instance, configuration, and sequence. Firstly, the large-scale deployment of microservices across numerous nodes (e.g., physical or virtual machines), leads to uncertainties in microservice communication. For instance, the microservice instances processing requests may be located in various network localities, resulting in inaccurate timeout estimates. Secondly, microservices are often configured in a decentralized manner, with different instances having different configurations. This leads to a high degree of variability in the behavior of microservices, making it challenging to identify patterns and relationships between services. Thirdly, the sequence in which microservices are executed can have a significant impact on the overall system behavior. Lastly, the high degree of inter-service dependency in microservice systems adds another layer of complexity to fault diagnosis. A fault in one microservice can propagate through the dependency graph, affecting other services and making it difficult to isolate the root cause of the problem.

Microservices often face system in practical scenarios, such as network latency and memory leaks, which may negatively affect their performance. Most of the data collected from microservices is stored in multi-variable time series, containing various key performance indicators of the microservices, such as request latency and CPU utilization. These usually reflect the system status, and these indicators record the status of different services in time series form [3]. Therefore, closely monitoring and analyzing various key performance indicators collected from each service instance, such as CPU load and network usage, has become the mainstream method for detecting and locating faults [4].

Recent research on micro-service system fault classification can be divided into multi-variable fault detection [5], and single-variable fault detection [6]. Single-variable detection methods are mainly based on a specific key performance indicator and can model time dependencies but cannot capture complex spatial relationships [5]. They are more likely to misidentify normal changes as anomalies, leading to more false alarms. In comparison, multi-variable fault detection methods can learn the inherent connections between microservices data. However, these methods are often not very effective, unable to fully capture the multi-scale features of data, and it is also challenging to model the complex relationships between different services, resulting in unsatisfactory classification results. For example, the classic Naive Bayes classifier [7] has certain biases when building models for related features, which may have a negative impact on anomaly detection results. GDN [8] network has certain advantages in building models for the complex relationships between different services in the microservice architecture. However, GDN still does not fully consider time features. In practical applications, temporal features are important for fault classification and prediction.

To address these issues, we propose an interpretable fault diagnosis framework tailored for microservices architecture. Specifically, since multi-scale neural transformations can enhance data diversity, and the execution sequence of microservices can represented

as a graph, we combine multi-scale neural transfromations with graph structure adjacency matrix. Then, the extracted spatiotemporal features and the topological structure characteristics of microservices are integrated into a multi-feature modeling, with the aimming to infer the relationships between microservices and achieve effective fault detection. Upon obtaining the corresponding multi-class fault classification results, we employ the PC algorithm [9] and PageRank algorithm [10] to diagnose and explore faults, thus explaining the potential causes of system failures. The main contributions of our work are as follows:

1) We propose an interpretable fault diagnosis framework for microservice architecture, namely Multi-scale Learnable Transformation Graph for Fault Classification and Diagnosis (MTG_CD). This approach combines multi-scale neural transformation and graph structure adjacency matrix, where we represent the order of executing microservices as a graph to extract structure-temporal features from system monitoring data, aiming to enhancing data diversity.
2) We employ graph convolutional networks (GCNs) to fuse the extracted spatio-temporal and microservice topology structure features in a multi-feature modeling approach. This helps to infer the relationships among microservices and achieve effective faults detection.
3) After obtaining the corresponding fault multi-classification results, we perform a coarse-grained level diagnosis and exploration to determine the underlying cause of system failures, which indicates that our model is interpretable.
4) Experiment results show that the MTG_CD model outperformed several baseline methods in the SockShop's microservice benchmark test, with an average macro F1 score improvement of 14.05%. The results demonstrate its superiority in detecting CPU usage overhead, memory leak, and network delay faults.

Related works

With the increasing complexity and scale of modern application systems, microservices have become a popular solution for enterprises to address these challenges. As a result, detecting and locating faults in microservice systems have become essential for ensuring system stability and reliability. Here, we divide related works into two main aspects: micro-service system fault classify detection and micro-service system fault diagnosis, respectively.

Micro-service system fault classify detection

In the field of micro-service system fault detection, a wide range of techniques have been proposed and widely applied. These techniques can be generally categorized into two major groups: machine learning methods, and deep learning methods.

Machine learning methods: They have been widely applied in various fields and have shown promising results. Some popular classification algorithms include Naive Bayes [7], Support Vector Machine (SVM) [11], Random Forest [12], K-Nearest Neighbors (KNN) [13] -based models, and others. For example, Murugan et al. [7] adopted a Naive Bayes classifier to model microservice event logs. By preprocessing and extracting features from log data, they classify normal and abnormal behaviors. Additionally, they use an adaptive learning method called AdaNet to dynamically adjust model parameters and improve detection accuracy. Russo et al. [11] utilized SVM to classify normal and abnormal data in microservice systems. To improve classification performance, they preprocessed and extracted features from the data. They also adopted cross-validation methods to evaluate model performance and adjust SVM hyperparameters for optimization. Miao et al. [12] employed the random forest algorithm to classify log data from microservice systems. They preprocessed the data and select features, then used random forests to classify normal and abnormal behaviors. To evaluate the performance of the model, they conducted a series of experiments and use cross-validation. Guan et al. [13] introduced a multi-view OVA model grounded on decision tree (MVDT) to facilitate the complexity of the decision tree structure and enhance the generalization capability for multi-classification tasks. Cinque et al. [14] adopted the KNN algorithm to classify normal and abnormal data in microservice systems. To improve classification performance, they also discussed how to select appropriate distance metrics and distance thresholds to enhance detection accuracy.

Deep learning based methods: Deep learning methods have gained significant attention in recent years due to their ability to automatically learn complex features and achieve state-of-the-art performance in various tasks. In the context of microservice anomaly detection, deep learning techniques have been applied to improve the accuracy and generalization ability of the models, such as neural networks [15], Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) [16], Graph-Based Methods [17]. For example, aiming to solve the problem of detecting potential anomalies in microservices, Hasnain et al. [15] used recurrent neural networks (RNN) based approach to capture and analyze temporal patterns in microservice logs, thereby detecting anomalies. Lindemann et al. [16] utilized long

short-term memory networks (LSTM) to capture temporal patterns and generate accurate predictions for microservice anomaly detection. Bae et al. [17] employed convolutional neural networks (CNN) for microservice anomaly detection to address issues related to accuracy, reliability, and real-time performance. CNN and LSTM models is employed by in DeepADNet [18] to classify multichannel EEG signals; [19] proposed a reinforcement learning-informed pattern mining framework for multivariate time series classification. A cooperative algorithm was proposed by Chen et al. [20] to automatically learn essential features and patterns in time series, which can be used for classification tasks; Zhao et al. [21] combines a multi-scale residual attention network (MSRA) and a generative adversarial network (GAN). It uses the MSRA network to extract features from hyperspectral images and enhances the model's generalization ability through data augmentation via the GAN network [4].

Some studies emplyed GNN. Aubet et al. [22] applied graph-theoretic methods to analyze the inter-service dependencies and detect anomalies based on the graph structures. Deng et al. employed graph structure based GDN [8] for binary classification models; Sha et al. [23] introduced a new semisupervised classification framework based on graph attention networks (GATs) for hyperspectral images (HSIs). Guillaume et al. [24] fused GCN and attention mechanisms to model multi-scale images, which enhanced the accuracy of multiclassification and system fault detection. Sheng et al. [25] considered employing the GCN for hyperspectral image classification, given its capability to perform convolutions on arbitrarily structured non-Euclidean data and its applicability to irregular image regions represented by graph topological information. Zhang et al. [26] discussed a flexible monitoring framework based on a dynamic-multilayer GCN that effectively captures temporal and spatial features from industrial time series data, in order to adapt to various tasks such as fault diagnosis and remaining useful life prediction. Wang et al. [27] presented a multivariate time series anomaly detection framework called Multiscale wavElet Graph AE (MEGA), which enhances anomaly detection accuracy by employing a dynamic graph module to capture changes in inter-variable dependencies.

However, the previously mentioned methods are unable to model the correlation and spatio-temporal characteristics of micro-service system fault features simultaneously, leading to limited feature learning. Moreover, for datasets with small sample sizes, extracting features becomes increasingly challenging. Therefore, it is necessary to design of multi- scale of feature extraction to enhance data diversity, so that improving the model's performance.

Micro-service system fault diagnosis methods

System fault diagnosis allows us to determine the underlying cause of anomalies among the various detected system faults. For instance, X. Zhou et al. [28] performed an industrial investigation to detect regular defects in microservice platforms, contemporary debugging methodologies, and the obstacles encountered by developers during implementation. Their research highlights the necessity of implementing intelligent trace examination that utilizes data-driven and learning-oriented strategies for trace comparison. X. Zhou et al. [29] executed an industrial investigation to detect common defects in microservice platforms, contemporary debugging strategies, and the difficulties encountered by developers during implementation. Their research underscores the necessity of adopting intelligent trace examination that utilizes data-driven and learning-oriented approaches for trace comparison. Ma et al. [30] focused on research on the challenge of identifying the root cause of exceptions in large-scale microservice frameworks, and introduced a technique referred to as ServiceRank. This approach ranks the services within the microservice architecture, enabling rapid identification of potential root causes of exceptions. Li et al. [31] presented Graph-Attention-Sage algorithm to categorizes and performs root cause analysis on anomalies by examining the graph neural network derived from dependency relationships among microservices. The TS-InvarNet method in [32] first extracts key performance indicator (KPI) sequences from the services by conducting time series analysis. Then, it aggregates and analyzes these KPI sequences in the spatial dimension, resulting in KPI invariants for each service node. Finally, TS-InvarNet employs machine learning algorithms to train an anomaly detection model utilizing these KPI invariants. Brandón et al. [33] introduced a root cause analysis framework that relies on graph representations of these architectures. These graphs allowed for comparing any abnormal situation occurring in the system with a library of anomalous graphs serving as a knowledge base for user troubleshooting. Xin et al. [2] proposed CausalRCA for fine-grained, automated, and real-time root cause localization. The method operates by employing a gradient-based causal structure learning approach to generate weighted causal graphs, followed by a root cause inference technique to identify root cause metrics. Liu et al. [34] investigated potential anomaly propagation chains based on dynamically generated service call graphs, and ranked potential root causes according to their correlation. Wu et al. [35] deduced root causes in real-time absence of any application detection, by correlating application performance symptoms with corresponding system resource utilization. Ma et al. [36] treated the system's components as individual nodes,

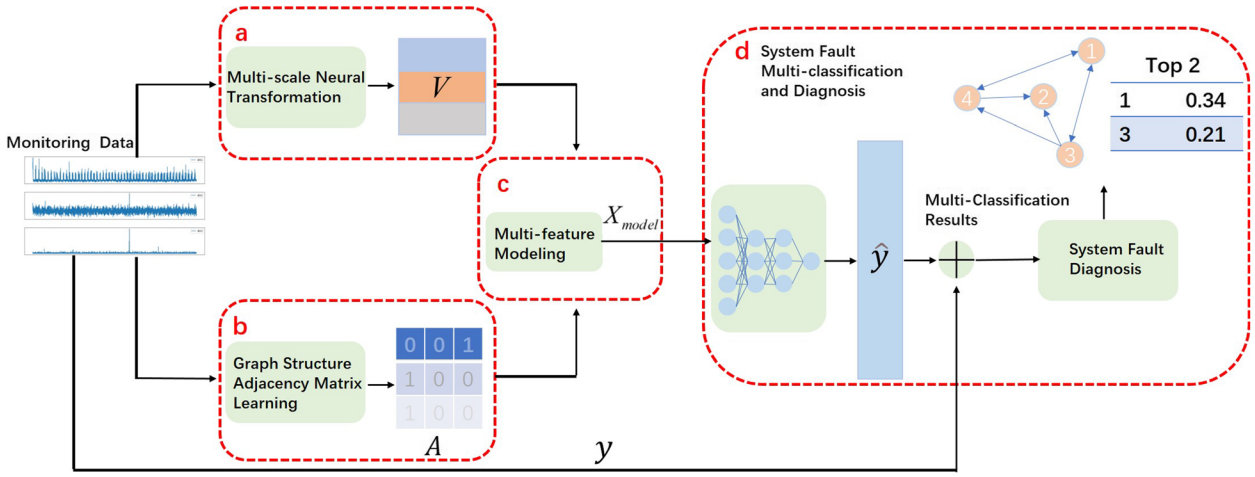

Fig. 2 The architecture of the proposed MTG_CD. **a** represents the Multi-scale Neural Transformation part; **b** represents the Graph Structure Adjacency Matrix Learning part; **c** represents the Multi-feature Modeling part; **d** represents the System Fault Multi-classification and Diagnosis part

and their interdependencies configure a graph. A graph neural network is trained, followed by the identification of the root cause utilizing the PC algorithm, and PageRank algorithm, where the PC [9] is a method based on probabilistic graphical models that infers causal relationships between variables by analyzing conditional independencies between them, and PageRank algorithm [10] determines the importance and ranking of web pages by analyzing the link relationships between web pages. Inspired by this, we employ the PageRank algorithm in this article to assess the impact of nodes on system faults.

System model

In this section, we first introduce the overall architecture of MTG_CD. After that, the four sub-modules of the MTG_CD are described, respectively.

Overall architecture of MTG_CD

Figure 2 shows an overview of the proposed MTG_CD architecture systematic fault multi-classification in microservices, MTG_CD consits four modules, including: (a) Multi-scale Neural Transformations, (b) Graph Structure Adjacency Matrix Learning, (c) Multi-feature Modeling, and (d) System Fault Multi-classification and Diagnosis, respectively. The general process of MTG_CD can be summarized as follows:

First of all, we collect and normalize data from the microservice fault monitoring system, where the collected data contain multiple attributes, such as order, payment, catalogue, user and carts, etc. Assuming the system fault data is derived from the real time monitoring of micro services, let $X = (x^1, ..., x^t, ..., x^T)^N \in R^{T \times N}$ be the input time series, where $t = 1, ..., T$. is the time step, and T is the total number of time steps. N is the feature dimensions of the data at each time step. In this paper, we employ the maximum-minimum normalization method to standardize the data and facilitate meaningful analysis.

Secondly, the normalized data are inputted into two modules simultaneously, namely (a) Multi-scale Neural Transformations and (b) Graph Structure Adjacency Matrix Learning. Regarding module (a), it enhances the diversity of the data through neural transformations. With respect to module (b), it helps to obtain the adjacency matrix of the graph.

Thirdly, the outputs from (a) Multi-scale Neural Transformations and (b) Graph Structure Adjacency Matrix Learning are simultaneously fed into the (c) Multi-feature Modeling section, to fuse the extracted spatio-temporal and microservice topology structure features in a multi-feature modeling approach. This helps to achieve effective faults detection.

Last but not least, the features captured by Multi-feature Modeling is inputted into the (d) System Fault Multi-classification and Diagnosis, which is beneficial to realize fault muti-classification and faults' causing analysis. The output vector $Y = (y^1, ..., y^t, ..., y^T)^M \in R^{T \times M}$ indicates the system fault multi-classification, where M is the number of system fault types, and $y^t = (0, 1, ..., M)$ represents whether the data at the t-th time step is an system fault. In actual scenarios, the dimensions of time series data may be time-varying, making it challenging to analyze and interpret the data effectively.

Multi-scale neural transformation

To enhance the diversity of the data in various scales, the (a) Multi-scale Neural Transformation is applied for fault multi-classification. The core of neural transformation technology is based on residual networks, which enhance

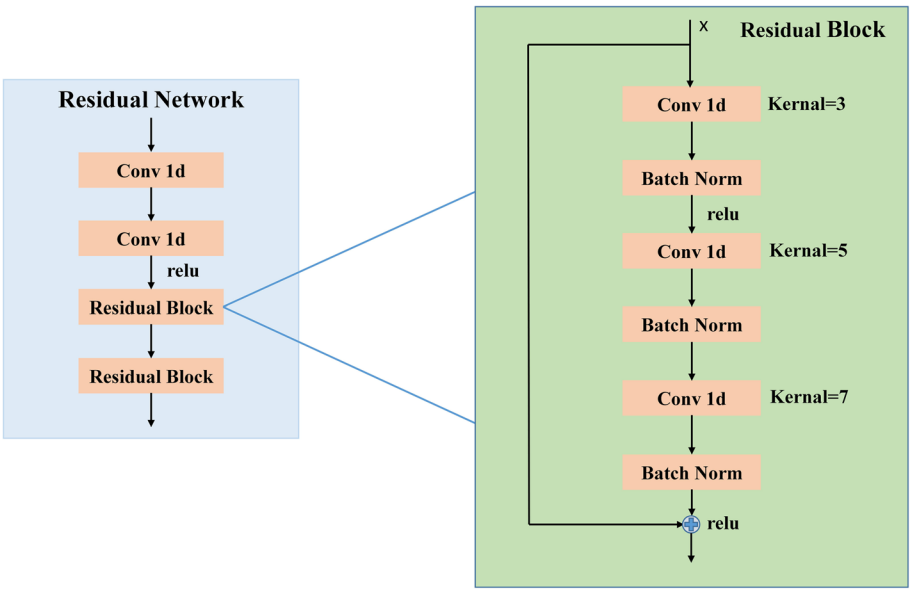

Fig. 3 One example of residual network containing numbers of residual blocks

the diversity of data features, including sparse anomalies, thus facilitating the model's ability to detect anomalies. On the other hand, the application of multi-scale techniques enables the model to capture information across various temporal scales and spatial dimensions. By learning the diverse characteristics of different anomalies, the model's generalization capability can be improved, as well as the accuracy of anomaly detection and multi-classification. As shown in Fig. 2a, Multi-scale Neural Transformation is a sub-module of proposed MTG_CD. We define M is the neural transformation function structure. as shown in Fig. 3, M is designed by a stack residual network containing numbers of residual blocks. Each residual block consists of several 1D convolutional layers, followed by instance normalization layers and ReLU activations.

Given the input micro-service system fault data X, the neural transformation result $V_k(X)$ is computed by Eq. (1) [37].

$$V_k(X) = M_k(X) + X \tag{1}$$

where k is the number of transformation.

Based on the characteristic of the neural transformation structure, the micro-service system fault data's temporal features can be captured. Specifically, the global and subtle temporal features can be get by the residual block, and local temporal features can be extracted by convolution operation. Both of the residual blocks and 1d convolutional layers improve the model's ability to model temporal features.

Graph structure adjacency matrix learning

In micro-service system, the data are graph-like structure data. To better process such data, we introduce Graph Structure Adjacency Matrix Learning to encode the correlation between micro-service system fault data and the adjacency matrix. In this paper, the graph generated by the adjacency matrix is used to describe the temporal-structural feature information of time series data. The adjacency matrix established in our work is established by two steps: first, calculating the Pearson correlation coefficient between the dimensions of the microservices system failure data. Then building the adjacency matrix based on the computed correlation. Therefore, the adjacency matrix reflects the correlation between different time series of the microservices system failure data, which is used to extract temporal-structural feature information to assist in system anomalies detection. In the adjacency matrix, the rows and columns denote the strength of the correlation between various time series. In other words, the larger value represents the stronger correlation, and vice versa. Assuming X is the input micro-service system fault data. The extracted adjacency matrix A can be defined as Eq. (2):

$$A = Adj(X) \tag{2}$$

where Adj is the adjacency matrix learning function. Pearson correlation coefficient is utilized to calculate the correlation among dimensions in micro-service system fault data. Subsequently, we set up our adjacency matrix based on the computed correlation, as shown in Eq. (3).

$$Adj = \frac{cov(x_i, y_i)}{\sigma x_i \sigma y_i} \tag{3}$$

where x_i represents the data in the i-th dimensions of the micro-service system fault data, $i = 1, ..., T$, while x_j is the data in the j-th dimensions of the micro-service system fault data, $j = 1, ..., T$, cov and σ are the covariance and standard deviation, respectively.

Multi-feature modeling

As mentioned above, Multi-scale Neural Transformation is used to extract the multi-scale temporal features, while Graph Structure Adjacency Matrix Learning is adopted to capture the structure-spatial feature information. The data outing from both Multi-scale Neural Transformation and Graph Structure Adjacency Matrix Learning are then input to the multi-feature Modeling, as shown in Fig. 2c.

Multi-feature Modeling is consisted by graph Convolutional Network (GCN) layer and a batch normalization layer. We employ the Multi-feature Modeling to model the input data with multiple features, including multi-scale temporal features and structure-spatial features. In particular, Multi-feature Modeling is capable of extracting information about the trend and periodic changes in data over time. Additionally, spatial features can reveal information about the spatial correlation between data points. By conducting a comprehensive analysis of both temporal and spatial characteristics, we can gain a deeper understanding of the data, uncover potential connections and rules, and enhance the model's performance. Furthermore, the modeled data comprises features that are advantageous for the multi-classification task of downstream system faults.

Let X_{model} be the output of Multi-feature Modeling, which is formulated in Eq. (4).

$$X_{model} = G(V, A) \tag{4}$$

where V is the multi-scale temporal features from the Multi-scale Neural Transformation, A is the structure-spatial feature information from the Graph Structure Adjacency Matrix Learning, G represents the Multi-feature Modeling combining by GCN layer and a batch normalization layer. Specifically, V and A perform matrix multiplication in GCN. The new feature matrix is obtained and multiplied by the GCN's weight matrix. The output is processed using an aggregation method and linear layer, resulting in the final output. The GCN layer can be expressed as Eq. (5).

$$h_i = \sigma\left(\sum_{j \in N(i)} \frac{1}{c_{ij}} W h_j\right) \tag{5}$$

where w and h_i represent the weight matrix and the feature vector of the i-th node, respectively. σ stands the activation function, and c_{ij} is a normalization constant that represents the elements of the i-th row and j-th column in adjacency matrix.

System fault multi-classification and diagnosis

To identify and distinguish different types of faults, thereby improving the reliability and stability of the system, we have established the system fault multi-classification and diagnosis module, as shown in Fig. 2d.

Firstly, the modeled feature vector X_{model} are mapped to specific prediction classes. Next, a standard multi-layer fully connected neural network is employed to convert the dimension of the feature vector to the number of classes. In addition, a cross-entropy loss function is adopted to compare the actual labels with the predicted labels. The cross-entropy loss function is defined in Eq. (6).

$$loss = -\frac{1}{M}\sum_{i}\sum_{j}^{N} y_{ij} log\left(\hat{y}_{ij}\right) \tag{6}$$

where M and N represent the number of training samples and the number of fault classification, respectively. while y and $\hat{y}$ is the actual label and predicted label, respectively.

Through this approach, we can more accurately predict potential fault types. After obtaining the results of classifying the fault data, we also conduct a coarse-grained level diagnosis and exploration to identify the root cause of such system faults. This involves tracing the microservices that are most likely to exhibit these faults. For implementing system fault diagnosis, we employ Principal Component (PC) and PageRank techniques to complement our analysis. By incorporating these two methods, we can further enhance our understanding of the underlying issues and contribute to the development of more efficient and reliable systems.

To be specific, we need to understand the degree of correlation between system faults and various microservices. In this process, we utilize the PC algorithm to find the DAG with minimum information loss in the initial G_0. This algorithm can retain critical information while reducing unnecessary redundancy, enabling us to analyze the relationship between system faults and microservices more precisely.

After finding an appropriate DAG, we perform a random walk using the PageRank algorithm. This algorithm calculates access probabilities based on the importance of nodes, helping us understand the relative importance of each node in the graph. By analyzing the importance of these nodes, we can identify the microservices that have the greatest impact on system faults.

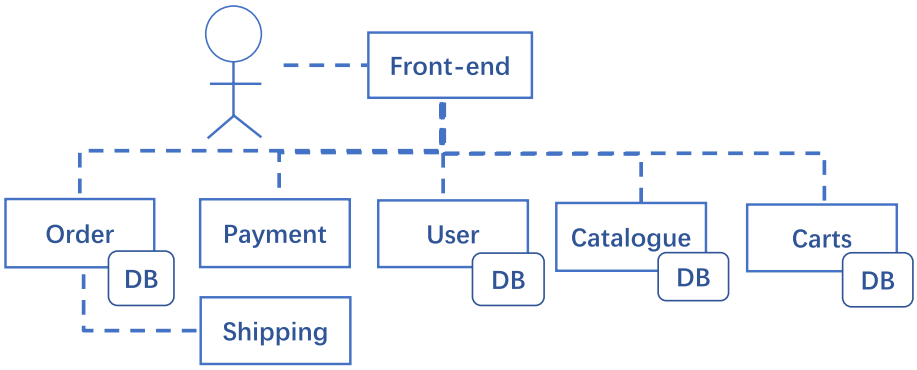

Fig. 4 The micro-service architecture of Sock Shop

Algorithm 1 is the process of system fault diagnosis based on multi-classification results. This algorithm takes multi-classified anomaly data as input and outputs the PageRank scores of each dimension after analyzing the causality graph. It is used to identify the most critical dimensions causing the anomalies, thus diagnosing the root cause of system faults. In summary, our method consists of two steps: first, using the PC algorithm in the initial G_0 to find the DAG with minimum information loss; second, after constructing the DAG related to system faults, employing the PageRank algorithm for a random walk and mining the microservices most likely to cause system faults based on node ranking. This approach helps us identify and resolve potential system faults more quickly, thereby improving the reliability and stability of the system.

Algorithm 1 System Fault Diagnosis based on multi-classification results

Input: Data X after anomaly multi-classification
Ouput: The *score* obtained after PageRank
1: Construct an initial causality graph G_0 based on the feature dimensions of the input data X;
2: The correlation coefficient matrix C between the feature dimensions is obtained using the PC algorithm;
3: Based on the correlation coefficient matrix C, the final causality graph G is constructed;
4: The PageRank algorithm performs a random walk on the causality graph G to get the PageRank *score* for each dimension;
5: Based on the PageRank *score*, filter the dimensions with high PageRank *score*, and output the *score*;

Evaluation experiments

In this section, we first introduce the dataset platform and experimental parameter settings for the microservices architecture. Then, we present the experimental results of our model and seven other comparative models. Finally, we show the system anomaly diagnosis experimental results.

Dataset

In order to conduct the evaluation experiment, we adopt a widely utilized microservices architecture testing platform, namely "Sock Shop", which comprises 13 core services[1] The primary focus of this research is on the following service domains: frontend presentation, product catalog, shopping cart, user management, order processing, payment functionality, and logistics services. As illustrated in Fig. 4, the microservices architecture of Sock Shop exhibits interconnected service modules, resulting in a higher complexity of the microservices system failure data we collected. This complexity also poses a challenge for our model in terms of multi-classification tasks.

The dataset contains spatial and temporal information of the microservices system. On one hand, the dataset includes service-level request latency metrics, as well as resource-level performance metrics, such as CPU utilization, memory utilization, disk read-write counts, and network send-receive bytes. This reflects the state of different services at different points in time. On the other hand, the dataset records the performance metrics of individual service instances within the microservices system, reflecting the spatial relationships between different service instances. Additionally, the document utilizes the dependencies between services to construct a graph structure that describes the spatial relationship information between services. In summary, the dataset presents a comprehensive view of the temporal and spatial information of the microservices system through time series and graph structures, providing important support for system fault detection and diagnosis.

[1] https://github.com/microservices-demo/microservices-demo

To simulate a real-world application environment, we inject three typical system faults: CPU profiling, memory leakage, and network latency [38]. In our microservices system fault multi-classification task, these three system faults are categorized into different classes, and are distinct from the normal data class. The application can normally run for 10 to 30 minutes before an anomaly occurs, and the injecting process of system faults is repeated at least five times for each system fault. It is worth mentioning that, each system fault lasts between 1 and 5 minutes.

In the experiment, we collect real-time data every 5 seconds, including both service-level and resource-level information. Specifically, we focus on the latency of each service at the service level, we collect performance metrics related to container resources at the resource level, such as CPU usage, memory usage, disk reads and writes, and network receive and transmit bytes. Table 1 summarizes the key characteristics of eight dataset. By the in-depth analysis of these data, we aim to provide a beneficial reference for microservices system fault diagnosis.

Experimental settings

our experimental platform is equipped with an on an server equipped with an Intel (R) Core (TM) i9-10900K CPU @ 3.70GHz, NVIDIA 2080Ti (12G) graphics card, and 32G RAM. The Python version installed on the server is 3.6, and the GPU-enabled PyTorch version is 1.4.0. We employ 3, 5, 7 as the convolutional kernel size in the M_k. We set initial learning rate and batch size as 0.00001 and 35, respectively. The dropout is set to 0.2, and categorical cross-entropy loss. We utilize the Adam optimizer to optimize the model's parameters and conduct 200 epochs of high-frequency training.

Evaluation metrics

We evaluate the performance of our proposed model and the baseline models using Four evaluation metrics, including macro-F1, macro-Precision, macro-Recall and macro-Acc.

Comparison with neural transformation *k*

Table 2 illustrates the impact of the number of neural transformation k on performance metrics under three different datasets: Catalogue, Shipping, and Payment. The K is set to 1, 3, 7, 12, 15, 17, and 19, respectively.

Table 2 The number of neural transformation on performance metrics under three different datasets: catalogue, shipping, and payment

	k	macro-F1	macro-Pre	macro-Rec	macro-Acc
Catalogue	1	0.8556	0.8494	0.8619	0.9494
	3	0.8732	0.8725	0.8745	0.9531
	7	0.8986	0.9362	0.8706	0.9663
	12	0.9056	0.9404	0.8761	0.9678
	15	0.9148	0.9426	0.8898	0.97
	17	0.9172	0.9491	0.8889	0.9714
	19	0.9029	0.9425	0.8711	0.9684
Shipping	1	0.836	0.8442	0.8292	0.9467
	3	0.8119	0.7942	0.8346	0.9336
	7	0.8668	0.882	0.8531	0.9552
	12	0.883	0.909	0.8613	0.9621
	15	0.9004	0.9183	0.8853	0.9667
	17	0.8979	0.9023	0.8938	0.9652
	19	0.8974	0.9063	0.8888	0.9667
Payment	1	0.9035	0.8988	0.9091	0.9675
	3	0.915	0.9292	0.9018	0.973
	7	0.9145	0.927	0.9031	0.9722
	12	0.9201	0.9313	0.9115	0.9737
	15	0.9426	0.9492	0.9369	0.9807
	17	0.9314	0.9406	0.923	0.9768
	19	0.9341	0.9391	0.9305	0.9776

The performance metrics include macro-F1, macro-Pre, and macro-Acc. We can observe that on the Catalogue dataset, the performance metrics generally improve with the increase of k, with the highest values being achieved at $k = 17$. Similarly, on the Shipping dataset, the performance metrics exhibit an upward trend as k increases, reaching their peak at $k = 15$. In the Payment dataset, the performance metrics also consistently improve with the increase of k, with the best performance being observed at $k = 15$. In general, k changes from 1 to 15, the performance of our model shows an upward trend, 15 to 17 shows a downward trend. This is because the k is too small, the model can not fully extract the relevant features of multivariate time series. The reason is that k is too large, the features extracted by the model are too redundant, which increases the workload to the following tasks and does

Table 1 The detail of partial datasets in the Sock Shop

Statistics	Carts	Catalogue	Frontend	Orders	Payment	Shipping	User
Dimension	35	35	36	35	36	35	36
Train size	3276	3229	3448	3273	3080	3096	3174
Test size	1405	1384	1479	1403	1321	1328	1361
Class	4	4	4	4	4	4	4

not use our anomaly detection. Setting the number of transformations allows the model to finely tune the feature extraction across various scales during the multi-scale transformation process. This precision helps the model to delve deeper into the intrinsic features of the data, which in turn enhances its ability to classify anomalies accurately. Thus, in the subsequent experiments, we will set k to 15.

Baselines

In addition to the proposed MTG_CD model, the following system fault multi-classification models, including HMM [39], NaiveBayes [7], Random Forset [11], Omni-Anomaly [5], CNN [16], TranAD [40], GDN [8] are compared to our proposed model on the Sock Shop testing platform.

Comparison with previous work

Table 3 presents the performance of our model and the baseline models on the selected eight datasets in terms of macro-F1, macro-Precision, macro-Recall and macro-Acc. The outstanding performance for each dataset is highlighted in bold.

It is noted that our model achieves the highest ranks in macro-F1, macro-Precision, macro-Recall, and macro-Accuracy. This demonstrates that our model is robust, consistently delivers solid performance in various data scenarios, and has strong generalization capabilities.

On the other hand, we also find that deep learning networks such as OmniAnomaly, TranADand GDN perform worse than classical, shallow network methods (e.g. NaiveBayes and RandomForest) in the microservice system fault multi-classification task, and may even exhibit counterproductive.

In the system fault multi-classification problem, different system faults may possess distinct feature representations. The OmniAnomaly algorithm may not fully consider the data distribution and diversity during the training process, which could lead to its performance degradation on certain datasets. The complex deep transformer network TranAD might not fully capture the differences and complexity between these categories, as the Transformer network primarily focuses on global dependencies in sequences and may not effectively capture local features of individual system fault categories. Similarly, graph-based networks GDN might have limited representational abilities for nodes and edges, failing to learn the feature differences between categories thoroughly. Consequently, these limitations lead to poor performance for both models.

The shallow neural network CNN has fewer parameters, which might not be sufficient to adapt to the complex system fault multi-classification problem. For this system fault multi-classification problem, more intricate models might be necessary to extract more abstract and advanced feature representations. For HMM, the reason for poor performance may be due to the large difference between the data distribution of the HMM model and the actual data distribution. In addition, during the training process, the model is also prone to falling into local optimal solutions. Classical methods like Naive Bayes and Random Forest outperform other baselines in handling system fault multi-classification problems, due to their ability to adapt to small samples and unbalanced data. However, since their limited feature representation capabilities, their performance falls low of our model.

Since macro-F1 is a more comprehensive evaluation metric, we separately compare the macro-F1 of different models across all datasets. The macro-F1 performance of various models on the eight datasets is presented in Fig. 5. Our model achieves the optimal macro-F1 in all datasets. This demonstrates that all components of our model function effectively and can learn data features better to achieve superior system fault multi-classification results.

Ablation experiment

To assess the effectiveness of each component in our model, we conducted ablation experiments on Catalogue, Shipping, and Payment datasets.

- w/o NT: replacing multi-scale neural transformation with multi-scale convolution.
- w/o MS: eliminating the multi-scale element in the multi-scale neural transformation
- w/o MNT: substituting multi-scale neural transformation directly with conventional convolution.

As illustrated in Table 4, the macro-F1 score decreases appropriately when we eliminate or substitute the corresponding components of the model. This evidence demonstrates that each component in our proposed model serves a distinct function and collectively promotes the successful accomplishment of the multi-classification task for microservice system faults.

Considering the performance indicators under three datasets, when the multi-scale component is removed, the average macro-F1 score decreases by 0.76%. This indicates that multi-scale effectively captures the feature information of abnormal data, thereby enhancing the model's capacity to detect random and scarce system faults.

Table 3 macro-F1, macro-Pre, macro-Rec and macro-Acc of the eight algorithms on eight datasets. The best performance is bolded

	Catalogue				Shipping				Orders				Carts			
Method	macro-F1	macro-Pre	macro-Rec	macro-Acc	macro-F1	macro-Pre	macro-Rec	macro-Acc	macro-F1	macro-Pre	macro-Rec	macro-Acc	macro-F1	macro-Pre	macro-Rec	macro-Acc
HMM	0.3108	0.3471	0.3373	0.5592	0.5154	0.5133	0.6288	0.7236	0.3877	0.3922	0.4931	0.5759	0.4051	0.4556	0.4857	0.4749
Naive-Bayes	0.8126	0.9285	0.7375	0.9443	0.5996	0.886	0.5261	0.9134	0.8662	0.8952	0.8403	0.9543	0.6063	0.6298	0.5911	0.9217
Random-Forest	0.6606	0.6982	0.6316	0.9436	0.8521	0.9012	0.8113	0.9518	0.6462	0.6759	0.6236	0.9337	0.7992	0.8629	0.7518	0.9451
CNN	0.5156	0.9069	0.4782	0.9062	0.4283	0.4679	0.4085	0.8957	0.6333	0.6841	0.5986	0.9314	0.4379	0.4568	0.4272	0.9028
Omni-Anomaly	0.2125	0.212	0.25	0.8692	0.2211	0.2142	0.238	0.8606	0.2315	0.2156	0.25	0.8622	0.2339	0.2197	0.25	0.879
TranAD	0.2383	0.2485	0.2521	0.8649	0.2313	0.2152	0.25	0.8607	0.2215	0.2056	0.25	0.8624	0.233	0.2188	0.25	0.868
GDN	0.2324	0.2172	0.25	0.8687	0.231	0.215	0.25	0.8609	0.232	0.2157	0.256	0.8627	0.4751	0.4525	0.5	0.905
Ours	**0.9148**	**0.9426**	**0.8898**	**0.97**	**0.9004**	**0.9183**	**0.8853**	**0.9667**	**0.9153**	**0.9114**	**0.9196**	**0.97**	**0.8862**	**0.8717**	**0.903**	**0.9636**
	Payment				User				Front-end				Vichalana			
Method	macro-F1	macro-Pre	macro-Rec	macro-Acc	macro-F1	macro-Pre	macro-Rec	macro-Acc	macro-F1	macro-Pre	macro-Rec	macro-Acc	macro-F1	macro-Pre	macro-Rec	macro-Acc
HMM	0.4426	0.487	0.5437	0.4169	0.3373	0.3815	0.4064	0.3668	0.4119	0.4582	0.4684	0.5814	0.2535	0.2409	0.3523	0.7079
Naive-Bayes	0.6888	0.6947	0.6847	0.9386	0.4148	0.4029	0.4276	0.9096	0.8213	0.8843	0.779	0.9371	0.3658	0.3623	0.3695	0.9432
Random-Forest	0.9053	0.9336	0.8805	0.9697	0.7406	**0.8602**	0.692	0.9441	0.8674	**0.8957**	0.8435	0.9519	0.4927	0.4913	0.4944	0.9506
CNN	0.6509	0.6811	0.6276	0.9196	0.421	0.4653	0.3982	0.912	0.6736	0.6945	0.6572	0.9312	0.454	0.4255	0.4983	0.9507
Omni-Anomaly	0.2013	0.2053	0.245	0.8614	0.2368	0.225	0.25	0.9	0.2293	0.2117	0.25	0.8471	0.1195	0.1165	0.1258	0.928
TranAD	0.2314	0.2154	0.25	0.8615	0.301	0.2954	0.3043	0.8765	0.2416	0.2417	0.2415	0.7295	0.1201	0.115	0.121	0.9277
GDN	0.2313	0.2141	0.25	0.8609	0.2369	0.2251	0.25	0.9004	0.2292	0.2117	0.25	0.8467	0.1203	0.1161	0.125	0.9288
Ours	**0.9426**	**0.9492**	**0.9369**	**0.9807**	**0.8213**	0.8167	**0.8265**	**0.9542**	**0.8759**	0.8522	**0.9063**	**0.9519**	**0.8318**	**0.8557**	**0.8142**	**0.9778**

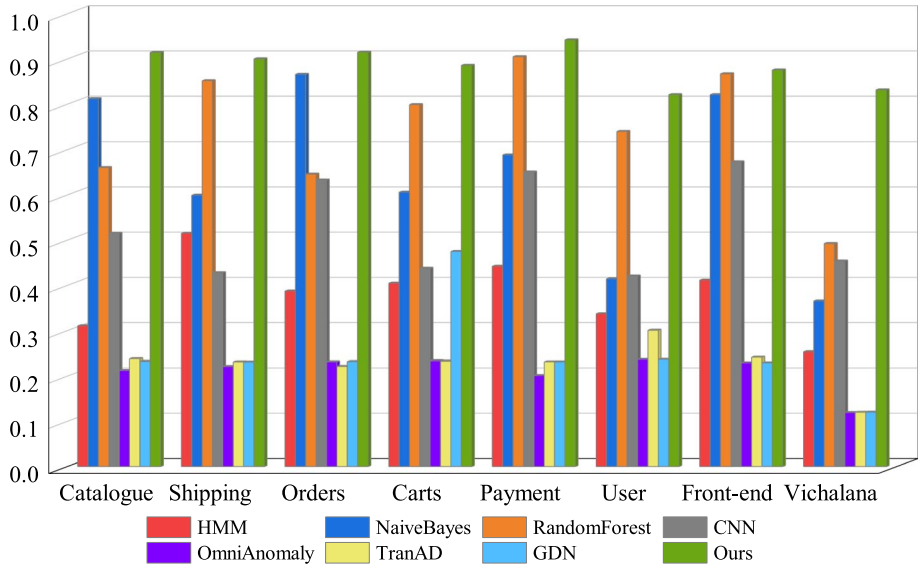

Fig. 5 Comparison of macro-F1 for eight models

Table 4 Ablation experiment on Catalogue, Shipping, and Payment datasets

	Method	macro-F1	macro-Pre	macro-Rec	macro-Acc
Catalogue	Ours	0.9148	0.9426	0.8898	0.97
	w/o NT	0.9029	0.9224	0.886	0.9663
	w/o MS	0.9123	0.9417	0.8885	0.9699
	w/o MNT	0.8972	0.9311	0.8691	0.9655
Shipping	Ours	0.9004	0.9183	0.8853	0.9667
	w/o NT	0.8683	0.8464	0.8929	0.9536
	w/o MS	0.8998	0.9051	0.8963	0.9665
	w/o MNT	0.8575	0.8532	0.8627	0.9521
Payment	Ours	0.9426	0.9492	0.9369	0.9807
	w/o NT	0.9146	0.9152	0.9154	0.9714
	w/o MS	0.923	0.926	0.9209	0.9745
	w/o MNT	0.9115	0.9127	0.9118	0.9699

When we replace the neural transformation with convolution, the average macro-F1 score decreases by 0.24%. This suggests that the neural transformation plays a vital role in enhancing data diversity by capturing diverse aspects of data features. This, in turn, enables the model to learn data more effectively and classify various data features.

Lastly, when we directly replace the multi-scale neural transformation with simple convolution, the average macro-F1 score decreases by 3.05%. This demonstrates that both components significantly contribute to extracting data features and classifying different data types, providing essential support for detecting microservice system faults.

In conclusion, each part of our model plays a crucial role in improving the model's performance and facilitating the successful completion of the multi-classification task for microservice system faults. The ablation experiments validate the effectiveness of the components in our proposed model and emphasize the importance of multi-scale neural transformation and multi-scale convolution in detecting and classifying microservice system faults.

System fault diagnosis results

In this subsection, we exemplify the effectiveness of our system fault diagnosis by selecting the Payment dataset and validating our method based on the classification results of the model. The causal propagation graph illustrating the interplay between microservices is presented in Fig. 6. The nodes in the figure represent seven microservices: 0 (Front-end), 1 (User), 2 (Catalogue), 3 (Orders), 4 (Carts), 5 (Payment), and 6 (Shipping). When a system fault occurs, it propagates through the connections between microservices, necessitating specific techniques to capture the causality between them.

To identify the most fundamental microservices underlying the system fault and diagnose its source, we utilize the PC algorithm to generate a causal propagation graph. This graph effectively displays the relationships between different microservices. Furthermore, we employ the PageRank algorithm to conduct a random walk in the causal propagation graph and calculate the abnormality

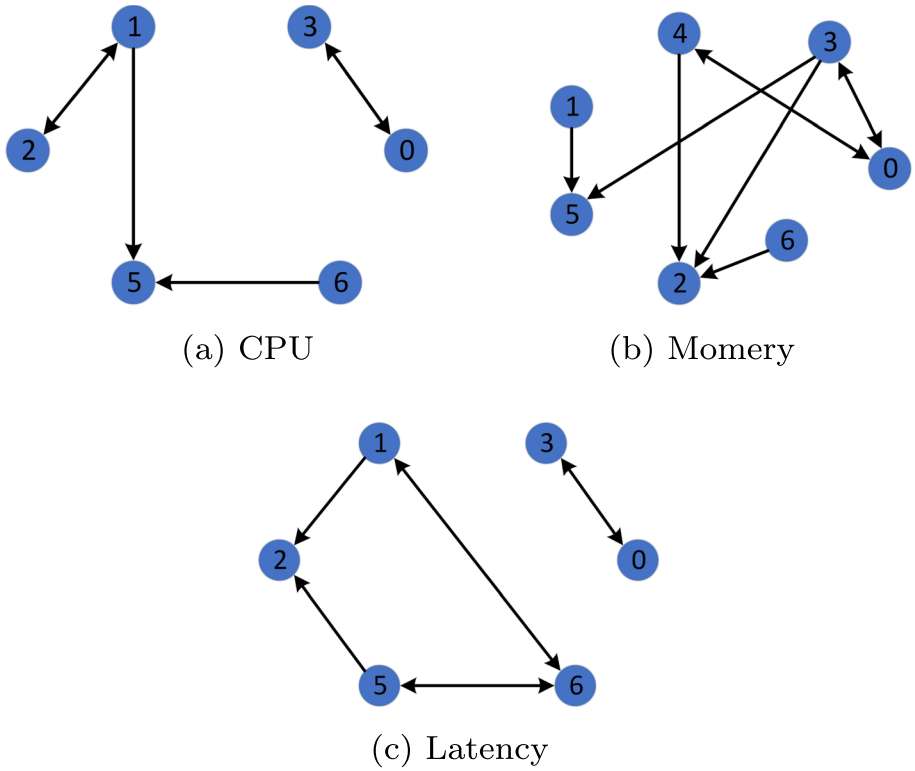

Fig. 6 Causal propagation diagram about CPU, Memory,and Latency system faults (treating different microservices as nodes)

score for each node. Finally, we select the top two nodes as the most fundamental causes of the system fault. The PageRank results are provided in Table 5.

Table 5 reveals that various microservices can contribute to different system faults, and the proportion of exceptions occurring for each microservice differs. For instance, concerning CPU system faults, the most likely culprits are User and Catalogue; for Memory system faults, it's Front-end and Orders; and for Latency system faults, Payment and Front-end are the most probable suspects. This indicates that our proposed approach can effectively determine the key microservices responsible for system faults, facilitating root cause diagnosis and enabling more targeted fault detection and optimization efforts.

It is worth noting that the presented study incorporates additional adjustments to the logical framework and includes supplementary explanations to ensure a more comprehensive and academic representation of the methodology and its applications. This comprehensive approach to system fault diagnosis in microservice systems can potentially benefit the development and maintenance of robust and efficient systems, contributing to the overall reliability of modern software systems.

Table 5 The results of PageRank

Anomaly	Top 2	
CPU	(2, 0.2217)	(1, 0.2145)
Memory	(0, 0.3147)	(3, 0.1722)
Latency	(5, 0.3083)	(0, 0.1666)

Conclusions and future work

In this paper, we study the problem of system faults potentially caused by real-time monitoring data in microservices. To classify and diagnose these system faults, we propose a supervised learning framework with a multi-scale approach to categorize occurring system faults and perform fault diagnosis based on the classified fault data.

Our proposed MTG_CD framework effectively addresses the challenge of robust real-time system fault identification in microservice applications. By utilizing graph structure adjacency matrix learning, multi-scale neural transformation, and graph convolutional networks, we achieve accurate and efficient fault diagnosis, paving the way for autonomous maintenance and repair in cloud-based microservice systems. Experimental results indicate that our model exhibits excellent performance, stability, and robustness.

Future research may proceed from two perspectives. First, We can conduct an in-depth investigation into the underlying causes of anomalies, extending our analysis beyond the service level. Second, we can optimize our model to achieve superior multi-classification performance for system faults. By doing so, we aim to better address various fault scenarios in microservice systems and enhance system reliability and stability.

Abbreviation

MTG_CD Multi-Scale Neural Transformation Graph

Acknowledgements

The authors would like to thank all the staf and students of school of computer and software engineering in Xihua university for contribution during this research process.

Authors' contributions

Problem formulation: Juan Chen, Rui Zhang. The proposed algorithm: Peng Chen, Jianhua Ren. Computer simulations: Yang Wang, Xi Li. Article preparation: Juan Chen, Zonging Wu, Ling Xiong. All authors have checked the manuscript and have agreed to the submission.

Funding

The work of this paper is supported by the Sichuan Province Science and Technology Program (2023JDRC0087).
Ministry of Education Program (HZKY20220578).

Availability of data and materials

No datasets were generated or analysed during the current study.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare no competing interests.

Received: 7 December 2023 Accepted: 5 May 2024
Published online: 15 May 2024

References

1. Al-Doghman F, Moustafa N, Khalil I, Sohrabi N, Tari Z, Zomaya AY (2023) Ai-enabled secure microservices in edge computing: Opportunities and challenges. IEEE Trans Serv Comput 16(2):1485–1504. https://doi.org/10.1109/TSC.2022.3155447
2. Xin R, Chen P, Zhao Z (2023) Causalrca: Causal inference based precise fine-grained root cause localization for microservice applications. J Syst Softw 203:111724. https://doi.org/10.1016/j.jss.2023.111724
3. Song Y, Xin R, Chen P, Zhang R, Chen J, Zhao Z (2023) Identifying performance anomalies in fluctuating cloud environments: A robust correlative-gnn-based explainable approach. Futur Gener Comput Syst 145:77–86. https://doi.org/10.1016/j.future.2023.03.020
4. Chen P, Liu H, Xin R, Carval T, Zhao J, Xia Y, Zhao Z (2022) Effectively Detecting Operational Anomalies In Large-Scale IoT Data Infrastructures By Using A GAN-Based Predictive Model. Comput J 65(11):2909–2925. https://doi.org/10.1093/comjnl/bxac085
5. Su Y, Zhao Y, Niu C et al (2019) Robust anomaly detection for multivariate time series through stochastic recurrent neural network[C]//Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, pp 2828–2837
6. Zhang L, Cheng W, Xing J, Chen X, Nie Z, Zhang S, Hong J, Xu Z (2023) Self-supervised variational graph autoencoder for system-level anomaly detection. IEEE Trans Instrum Meas 72:1–11. https://doi.org/10.1109/TIM.2023.3323989
7. Murugan K, Suresh P (2018) Efficient anomaly intrusion detection using hybrid probabilistic techniques in wireless ad hoc network. Int J Netw Secur 20:730–737
8. Deng A, Hooi B (2021) Graph neural network-based anomaly detection in multivariate time series. ArXiv abs/2106.06947. http://arxiv.org/abs/2106.06947v1
9. Le TD, Hoang T, Li J, Liu L, Liu H, Hu S (2019) A fast pc algorithm for high dimensional causal discovery with multi-core pcs. IEEE/ACM Trans Comput Biol Bioinforma 16(5):1483–1495. https://doi.org/10.1109/TCBB.2016.2591526
10. Elbarougy R, Behery G, Khatib AE (2020) Extractive arabic text summarization using modified pagerank algorithm. Egypt Inform J 21:73–81
11. Russo L, Sarda K, Glielmo L, Acernese A (2021) Fault detection and diagnosis in steel industry: a one class-support vector machine approach. In: 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). pp 2304–2309. https://doi.org/10.1109/SMC52423.2021.9659069
12. HU M, WANG K (2019) Random forest based on double features and relaxation boundary for anomaly detection. J Comput Appl 39(4):956
13. Guan X, Liang J, Qian Y, Pang J (2017) A multi-view ova model based on decision tree for multi-classification tasks. Knowl Based Syst 138:208–219. https://doi.org/10.1016/j.knosys.2017.10.004
14. Cinque M, Corte RD, Pecchia A (2022) Micro2vec: Anomaly detection in microservices systems by mining numeric representations of computer logs. J Netw Comput Appl 208:103515
15. Hasnain M, Jeong SR, Pasha MF, Ghani I (2020) Performance anomaly detection in web services: An rnn- based approach using dynamic quality of service features. Comput Mater Continua 64(2):729–752. https://doi.org/10.32604/cmc.2020.010394
16. Lindemann B, Maschler B, Sahlab N, Weyrich M (2021) A survey on anomaly detection for technical systems using lstm networks. Comput Ind 131:103498
17. Bae J, Jung W, Park Y-H (2022) Normal data based rotating machine anomaly detection using cnn with self-labeling. Smart Struct Syst 29(6):757–766
18. Ho TKK, Jeon Y, Na E, Ullah Z, Kim BC, Lee KH, Song JI, Gwak J (2021) Deepadnet: A cnn-lstm model for the multi-class classification of alzheimer's disease using multichannel eeg. Alzheimers Dement 17:e057573
19. Gao G, Gao Q, Yang X, Pajic M, Chi M (2022) A reinforcement learning-informed pattern mining framework for multivariate time series classification. In: In the Proceeding of 31th International Joint Conference on Artificial Intelligence (IJCAI-22)
20. Chen J, Chen P, Niu X, Wu Z, Xiong L, Shi C (2022) Task offloading in hybrid-decision-based multi-cloud computing network: a cooperative multi-agent deep reinforcement learning. J Cloud Comput 11(1):1–17
21. Zhao J, Hu L, Huang L, Wang C, Liang D (2023) Msra-g: Combination of multi-scale residual attention network and generative adversarial networks for hyperspectral image classification. Eng Appl Artif Intell 121:106017
22. Aubet FX, Pahl MO, Liebald S, Norouzian MR (2018) Graph-based anomaly detection for iot microservices. Measurements 120(140):160. https://doi.org/10.13140/RG.2.2.22381.69609
23. Sha A, Wang B, Wu X, Zhang L (2020) Semisupervised classification for hyperspectral images using graph attention networks. IEEE Geosci Remote Sens Lett 18(1):157–161
24. Pelluet G, Rizkallah M, Tardy M, Acosta O, Mateus D (2022) Multi-scale graph neural networks for mammography classification and abnormality detection. In: Annual Conference on Medical Image Understanding and Analysis. Springer International Publishing, Cham, p 636–650
25. Wan S, Gong C, Zhong P, Du B, Zhang L, Yang J (2019) Multiscale dynamic graph convolutional network for hyperspectral image classification. IEEE Trans Geosci Remote Sens 58(5):3162–3177
26. Zhang XJ, Ding X, Zhang HF, Pan DH, Zhong K (2023) A flexible monitoring framework via dynamic-multilayer graph convolution network. IEEE Trans Instrum Meas 72:1–11. https://doi.org/10.1109/TIM.2023.3284956

27. Wang J, Shao S, Bai Y, Deng J, Lin Y (2023) Multiscale wavelet graph autoencoder for multivariate time-series anomaly detection. IEEE Trans Instrum Meas 72:1–11. https://doi.org/10.1109/TIM.2022.3223142
28. Zhou X, Peng X, Xie T, Sun J, Ji C, Li W, Ding D (2021) Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. IEEE Trans Softw Eng 47(2):243–260. https://doi.org/10.1109/TSE.2018.2887384
29. Zhou X, Peng X, Xie T, Sun J, Ji C, Li W, Ding D (2022) Delta debugging microservice systems with parallel optimization. IEEE Trans Serv Comput 15(1):16–29. https://doi.org/10.1109/TSC.2019.2919823
30. Ma M, Lin W, Pan D, Wang P (2022) Servicerank: Root cause identification of anomaly in large-scale microservice architectures. IEEE Trans Dependable Secure Comput 19(5):3087–3100. https://doi.org/10.1109/TDSC.2021.3083671
31. Li Z, Tu Y, Ma Z (2022) Root cause analysis of anomalies based on graph convolutional neural network. Int J Softw Eng Knowl Eng 32(08):1155–1177. https://doi.org/10.1142/S0218194022500395
32. Chen P, Qi Y, Hou D (2017) Invarnet-x: A black-box invariant-based approach to diagnosing big data systems. IEEE Trans Emerg Top Comput 5(4):450–465. https://doi.org/10.1109/TETC.2015.2497143
33. Brandón Álvaro, Solé M, Huélamo A, Solans D, Pérez MS, Muntés-Mulero V (2020) Graph-based root cause analysis for service-oriented and microservice architectures. J Syst Softw 159:110432. https://doi.org/10.1016/j.jss.2019.110432
34. Liu D, He C, Peng X, Lin FF, Zhang C, Gong S, Li Z, Ou J, Wu Z (2021) Microhecl: High-efficient root cause localization in large-scale microservice systems. https://arxiv.org/abs/2103.01782
35. Wu L, Tordsson J, Elmroth E, Kao O (2020) Microrca: Root cause localization of performance issues in microservices. In: NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium. pp 1–9. https://doi.org/10.1109/NOMS47738.2020.9110353
36. Ma M, Xu J, Wang Y et al (2020) AutoMAP: Diagnose your Microservice-based web applications automatically. WWW '20: The Web Conference 2020. https://doi.org/10.1145/3366423.3380111
37. Qiu C, Pfrommer T, Kloft M, Mandt S, Rudolph MR (2021) Neural transformation learning for deep anomaly detection beyond images. ArXiv abs/2103.16440. https://arxiv.org/abs/2103.16440v1
38. Mariani L, Monni C, Pezzé M, Riganelli O, Xin R (2018) Localizing faults in cloud systems. In: 2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST). pp 262–273. https://doi.org/10.1109/ICST.2018.00034
39. Fouad MA, Abdel-Hamid AT (2019) On detecting iot power signature anomalies using hidden markov model (hmm). In: 2019 31st International Conference on Microelectronics (ICM). pp 108–112. https://doi.org/10.1109/ICM48031.2019.9021483
40. Tuli S, Casale G, Jennings NR (2022) Tranad: Deep transformer networks for anomaly detection in multivariate time series data. CoRR abs/2201.07284. https://doi.org/10.14778/3514061.3514067

Publisher's Note